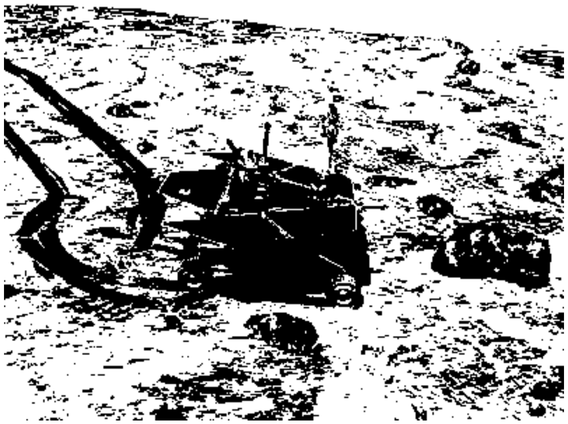


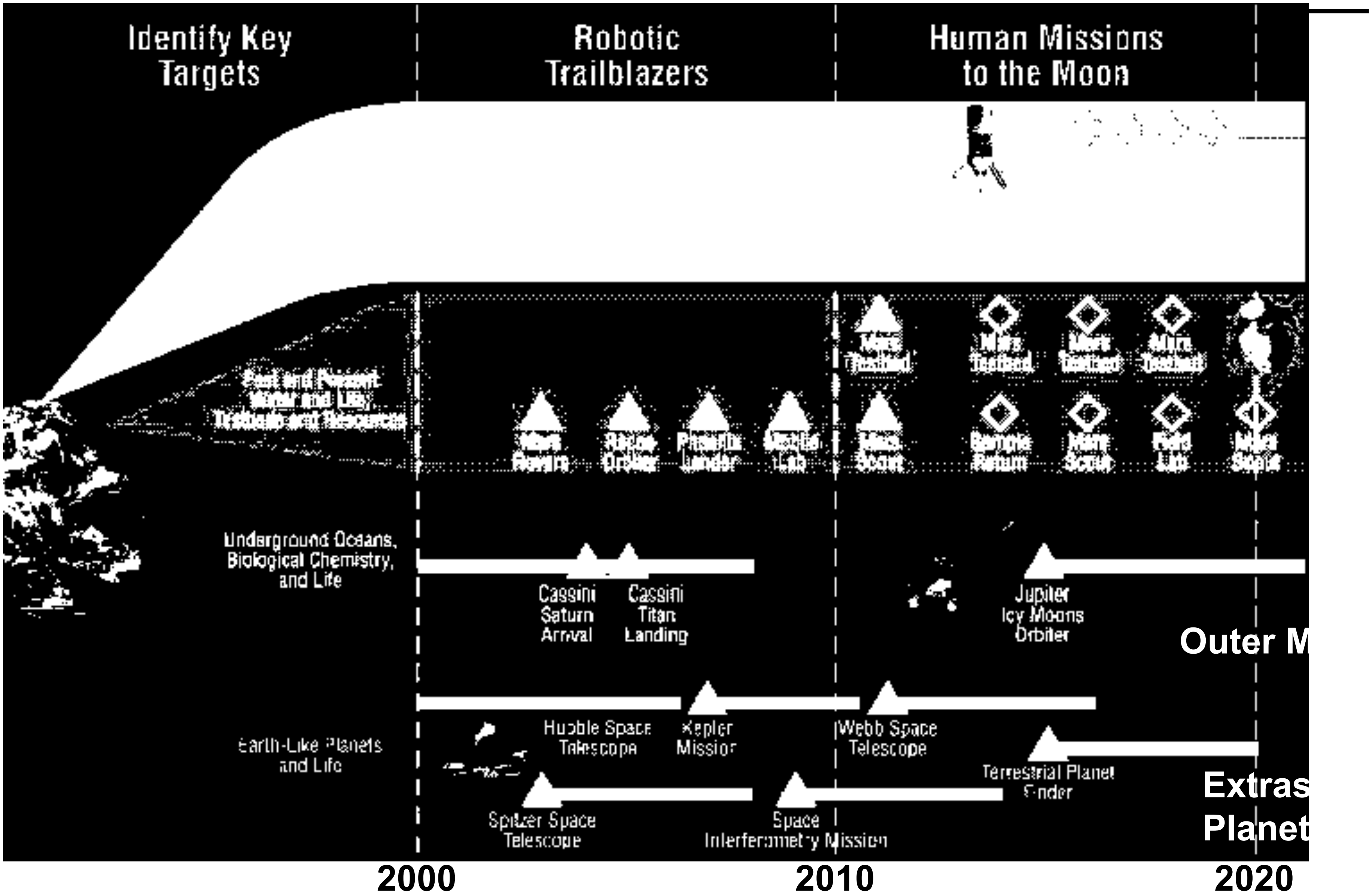
G. Brat
USRA/RIACS

National Aeronautics and
Space Administration

The Vision for Space Exploration

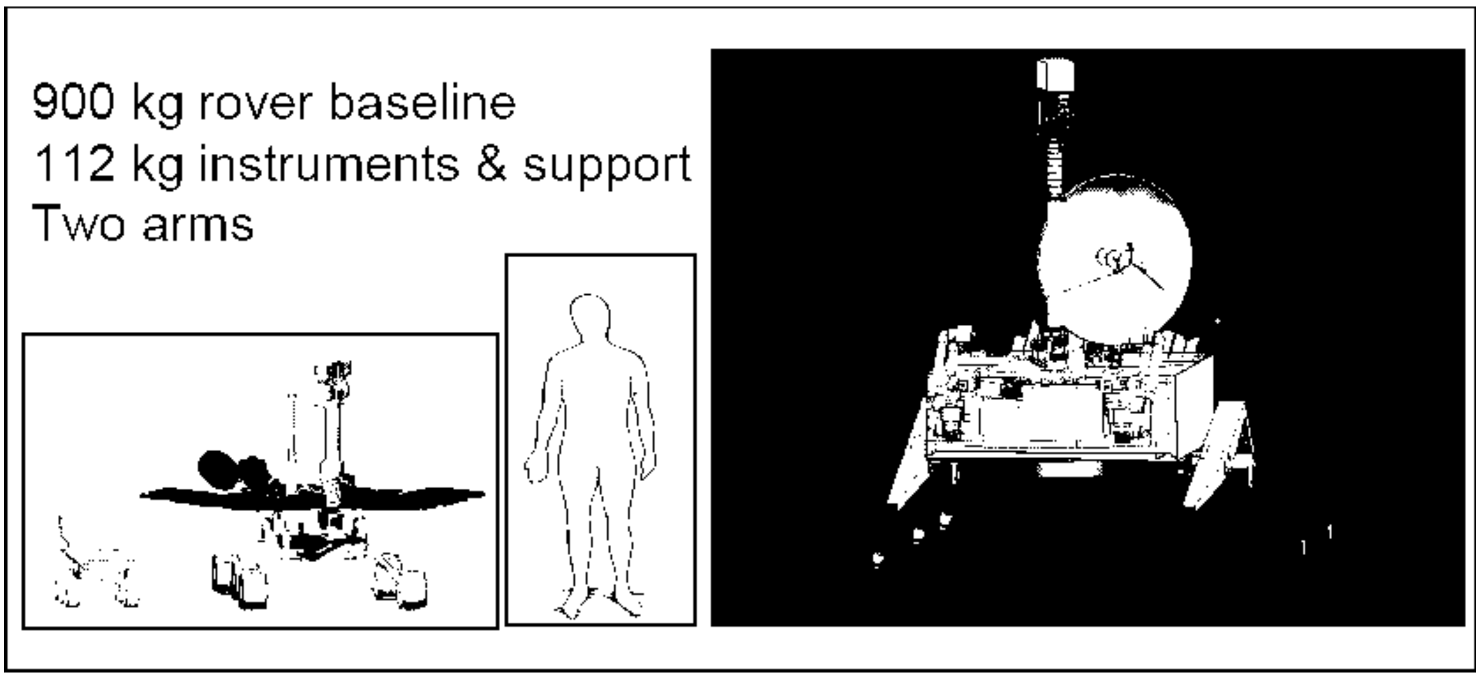
February 2004

- Implement a sustained and affordable human and robotic program to explore the solar system and beyond;
- Extend human presence across the solar system, starting with a return to the Moon by the year 2020, in preparation of the exploration of Mars and other destination;
- Develop the innovative technologies, knowledge, and infrastructures, both to explore and to support decisions about the destinations for human exploration;
- Promote international and commercial participation in exploration to further U.S. scientific, security, and economic interests.

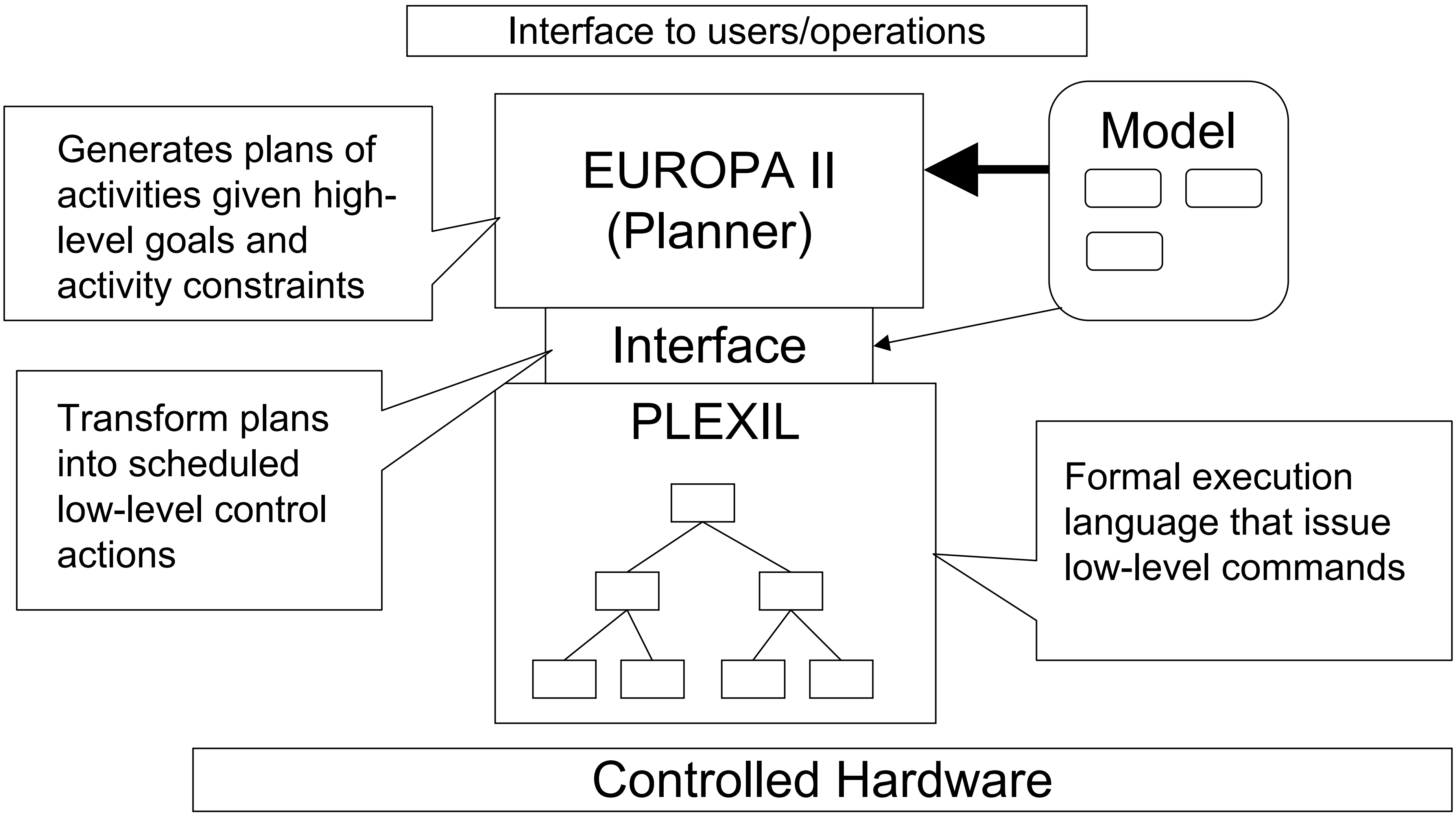


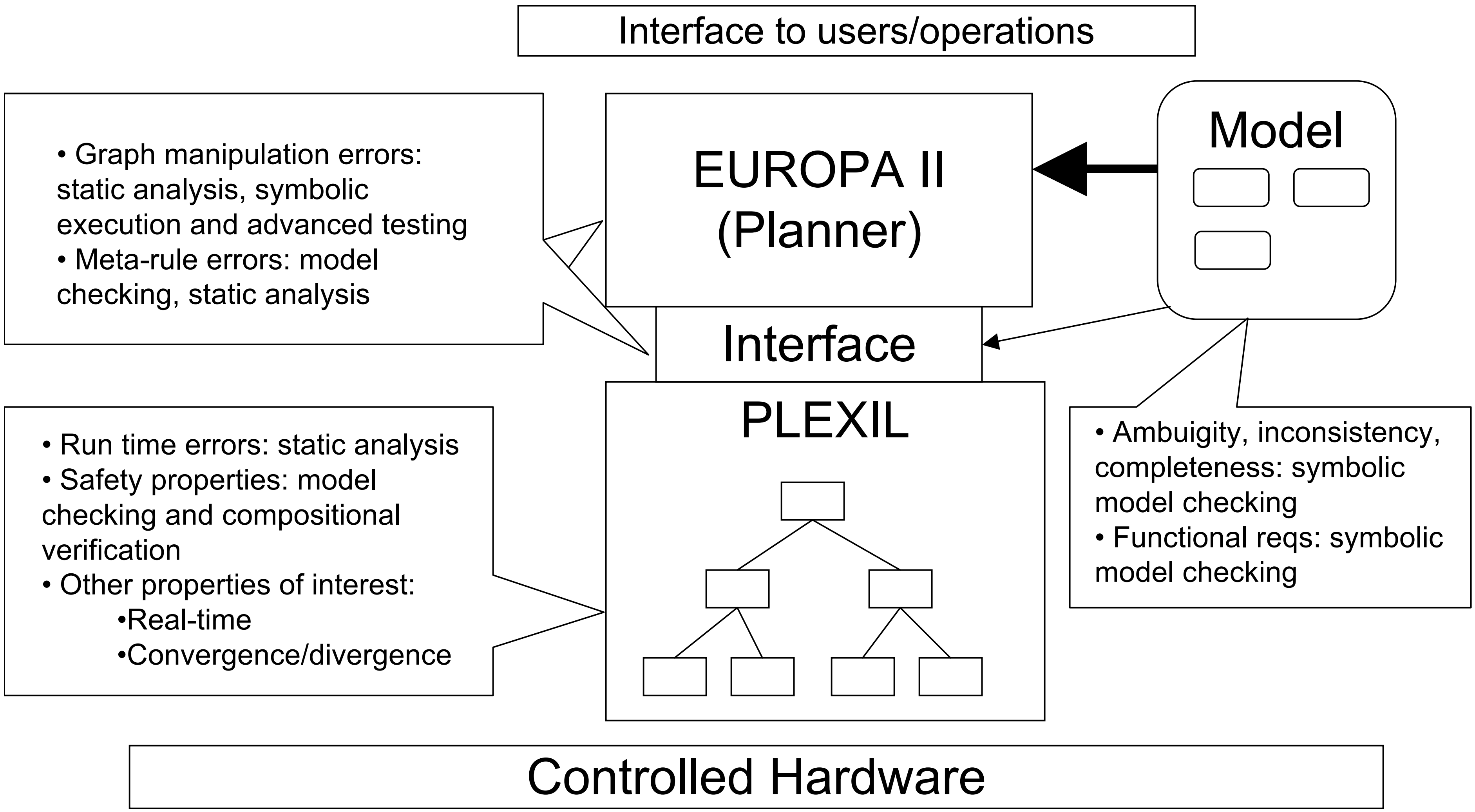
Identify Key Targets
Robotic Trailblazers
Human Missions to the Moon
Underground Oceans, Biological Chemistry, and Life
Cassini Saturn Arrival
Cassini Titan Landing
Jupiter Icy Moons Orbiter
Outer M
Earth-Like Planets and Life
Hubble Space Telescope
Kepler Mission
Webb Space Telescope
Terrestrial Planet Finder
Spitzer Space Telescope
Space Interferometry Mission
Extras
Planet
2000
2010
2020

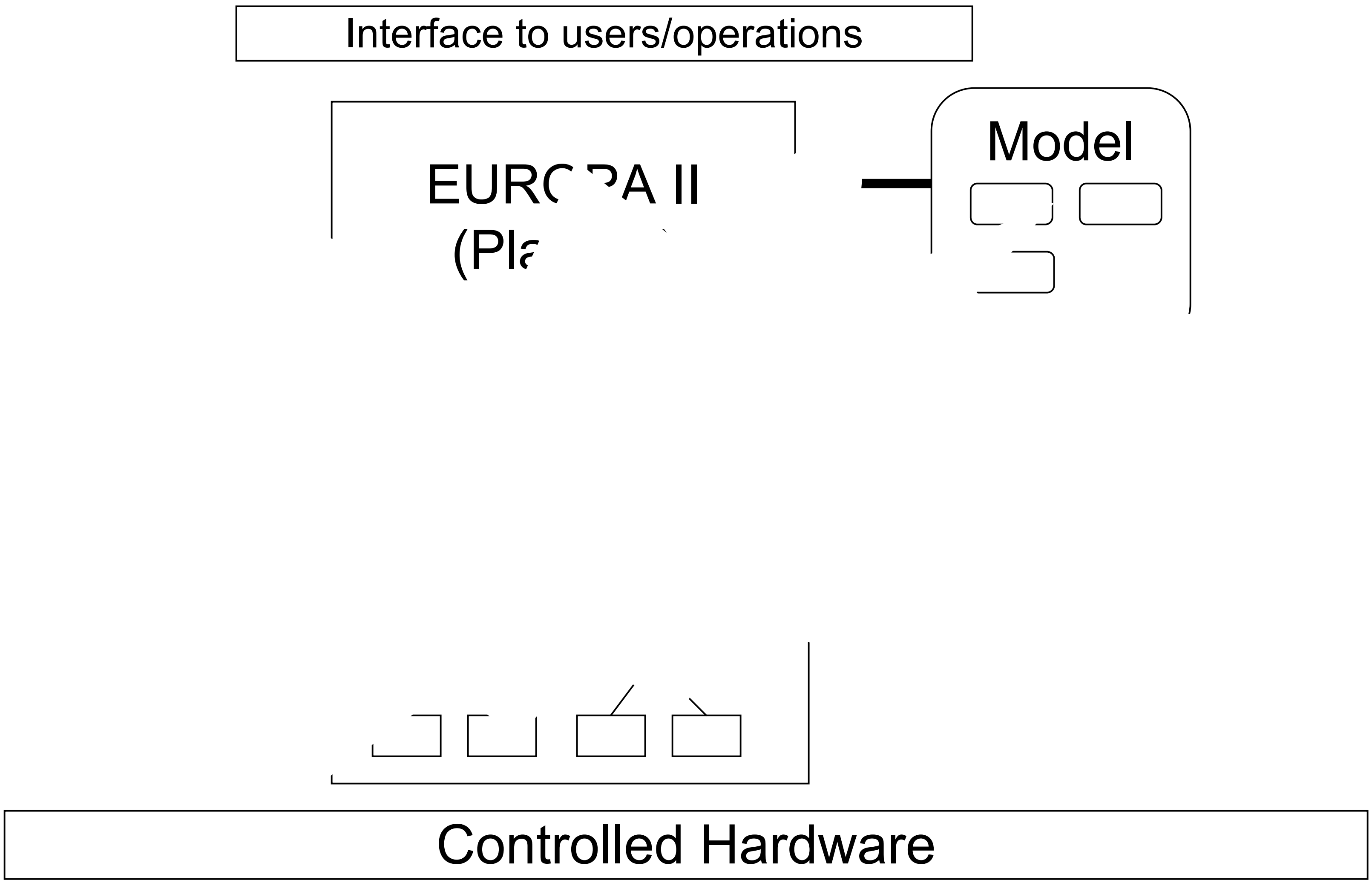
- Mission:
  - Long range traverses (< 6km)
  - Collect samples
  - Analyze samples on-board



- Need to develop three systems for each mission:
  - Flight software
  - Ground software
  - Simulation software
- Flight software
  - Rovers will require more adaptable software to do long traverses for example
- Ground software
  - Need planning software for planning operations
  - Need autonomous execution for uploading and executing commands on ISS or on-orbit
- V&V of a different type of software systems

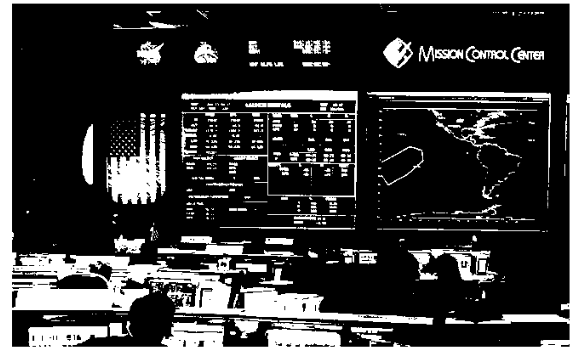

Interface to users/operations
Generates plans of activities given high-level goals and activity constraints
EUROPA II (Planner)
Model
Interface
Transform plans into scheduled low-level control actions
PLEXIL
Formal execution language that issue low-level commands
Controlled Hardware

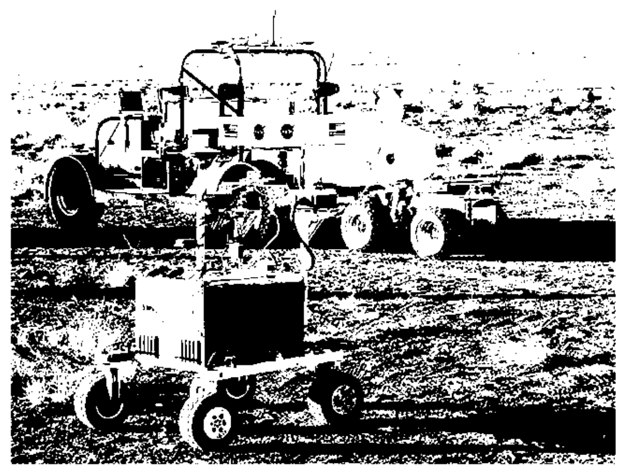

Interface to users/operations
• Graph manipulation errors: static analysis, symbolic execution and advanced testing
• Meta-rule errors: model checking, static analysis
EUROPA II (Planner)
Model
Interface
PLEXIL
• Run time errors: static analysis
• Safety properties: model checking and compositional verification
• Other properties of interest:
•Real-time
•Convergence/divergence
• Ambuigity, inconsistency, completeness: symbolic model checking
• Functional reqs: symbolic model checking
Controlled Hardware

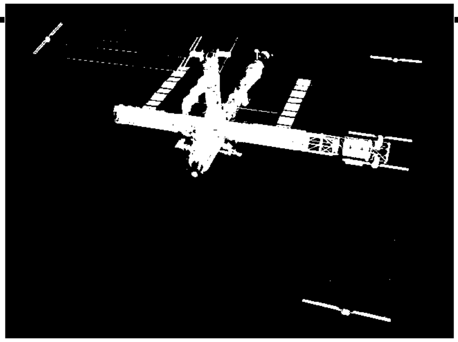


Interface to users/operations
EUROPA II
Model
Controlled Hardware

- Autonomy for Operations
  - PIs: Jeremy Frank & Ari Jonsson
  - PM: Robert Brummett
- Project goal:
  - Develop and mature needed automation software
  - capabilities for Constellation mission operations, onboard
  - control, crew assistance and robotics.
- Core capabilities
  - Human in-the-loop automation
  - Monitored execution
  - Decision support
  - Operation requirement studies
  - Simulation and testbeds
  - Application and prototypes
  - Verification





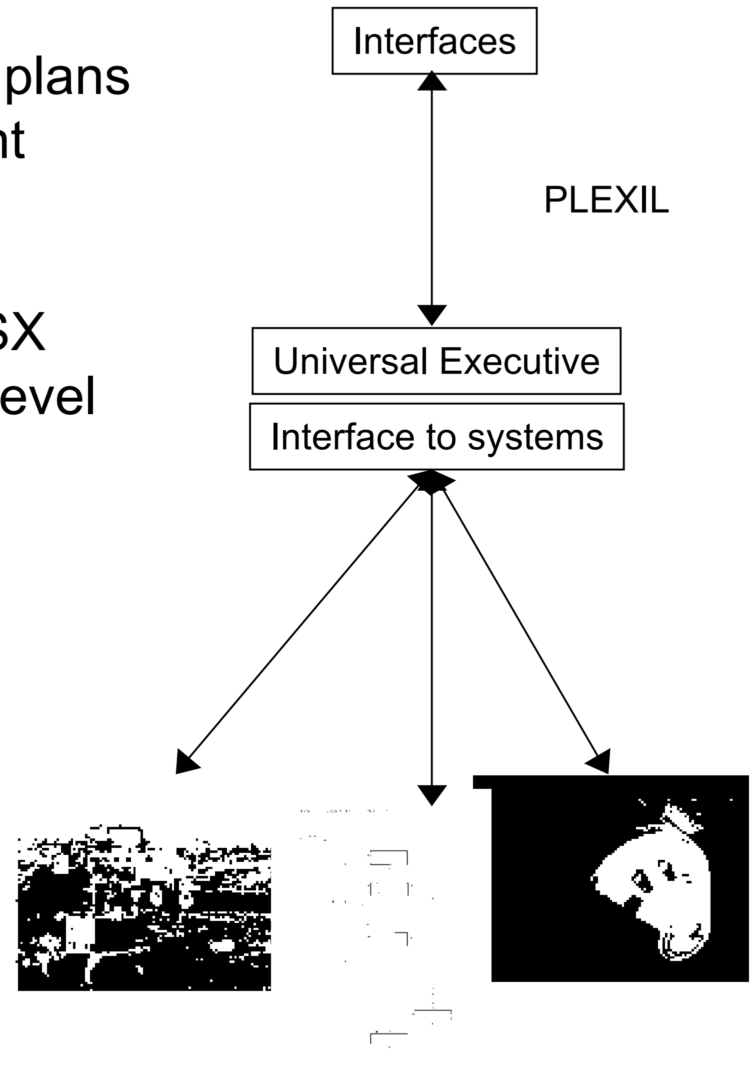


- **Mission Operations**
  - Operating procedure generation
  - Space flight operations planning
  - Remote system operations (nominal and off-nominal)
  - Support of crew control (nominal and off-nominal)
- **Crewed Spacecraft Operations**
  - Spacecraft systems operations (nominal and off-nominal)
- **Robotic Operations**
  - Explorers and scouts on the lunar surface
  - Assistants and tools for human explorers
- **Lunar Infrastructure Operations**
  - Control of habitats, communications and power equipment, etc.
- **Unmanned Spacecraft Operations**
  - Remote system operations (nominal and off-nominal)

- **Mission Operations**
  - State of art : Many tools, lack of interoperability
  - Need: mission operations paradigm
- **Crewed Spacecraft Operations**
  - State of art : Crew relies on ground to support and control operations
  - Need: Crews able to operate systems and own tasks
- **Robotics Operations**
  - State of art: Requires multiple operators for command and monitoring
  - Need: robot operations
- **Lunar Surface Operations**
  - State of art : Ground-based operation of most surface assets
  - Need: robot operations
- **Unmanned Spacecraft Operations**
  - State of art: Requires direct human command and monitoring
  - Need: operations

- **Key elements of technology**
  - Re-usable, interoperable and adaptable architecture
    - **Data-driven general and re-usable modules**
    - **Common data specifications support adaptability, evolvability and interoperability of tools based on standards developed by CSI**
  - Automation capabilities
    - **Monitoring and analysis of telemetry and system states**
    - **: From help for users to on-board decision-making**
    - **: Carry out decisions and plans, from humans and automation**
  - Human interaction support
    - **Adjustable automation allows humans to handle more or less as needed**
    - **Assistance provides summary of information, options, evaluations, warnings**
    - **Complementary capabilities based on computational power**
- **Flexible and reusable - on ground and on board**
  - Enable transition from initial manual flights to sustainable operations
  - Same core capabilities used on ground, in flight and on lunar surface

- **Executive**
  - Lightweight engine for executing PLEXIL plans
    - Small memory and processor footprint
  - General and reusable
    - Same engine for many applications
  - Compiles on VxWorks, Linux, Solaris, OSX
    - Simple, well defined interface to low level control
  - Commanding interface
    - Sensing interface
  - Provides tools for users
    -



- **Applications**
  - Drives procedure execution automation
  - Executes plans for on-board operations
  - Runs K10 rover activity plans on board

- **Procedures**
  - Notion generalizes a number of existing concepts:
  Command sequences, plans, checklists, diagnosis procedures, etc.
- **Procedures for both humans and automation**
  - : Human-understandable; e.g., operations procedures
  - : Machine-understandable; e.g., plans and command sequences
  - Need a combination to enable adjustable automation
- **Procedure Representation Language (PRL)**
  - Combines ISS procedure schema with PLEXIL schema
  - XML-based language
- **Elements of PRL**
  - **Meta data** provides names, context, version, etc. for procedure
  - **Control data** provides logical control and safety conditions
  - **Steps and nodes** structure procedure for human readability
  - **Instructions** specify instructions, commands, etc.

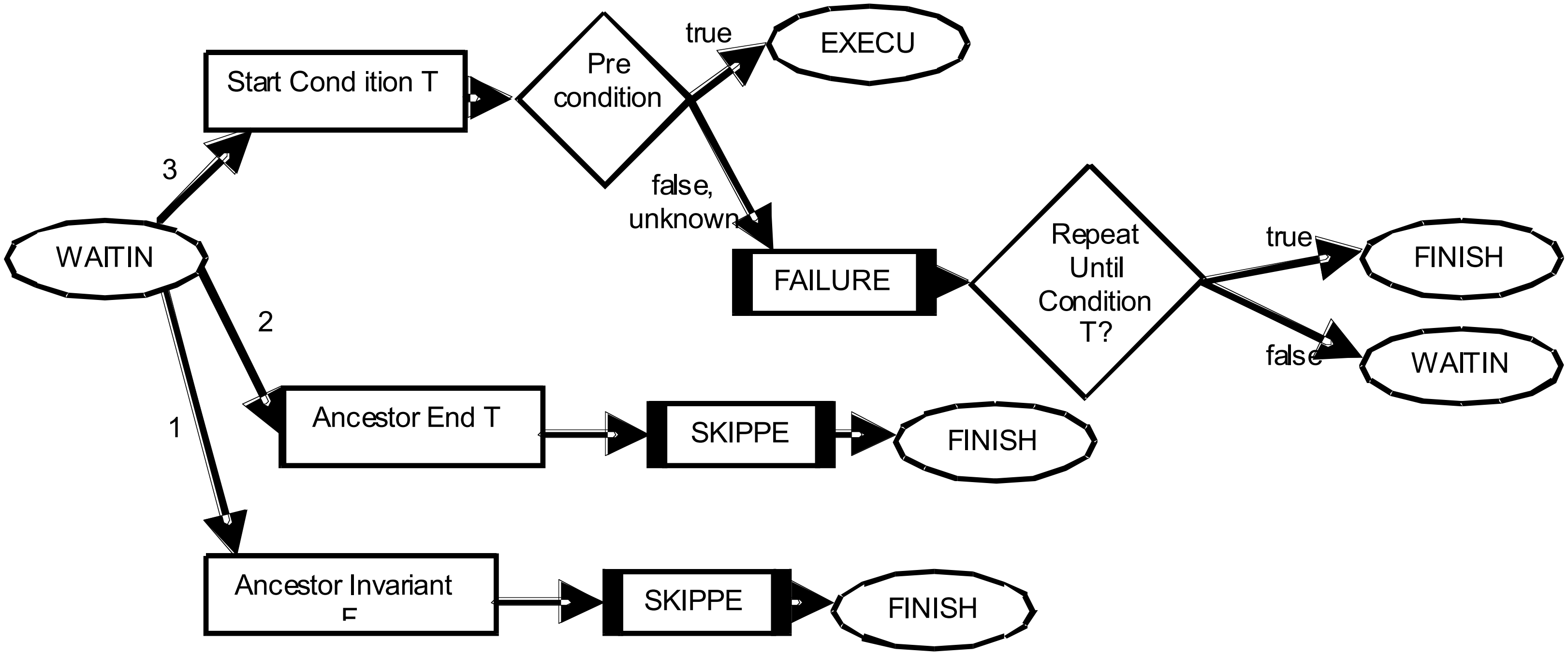
- Main focus: how to validate procedures?
- We have five major efforts under way
  - Definition of formal semantics of PLEXIL language
  - Model-based generation of test plans for PLEXIL
  - Model checking of PLEXIL procedures
  - Simulation of PRL procedures
  - Model checking of PRL procedures

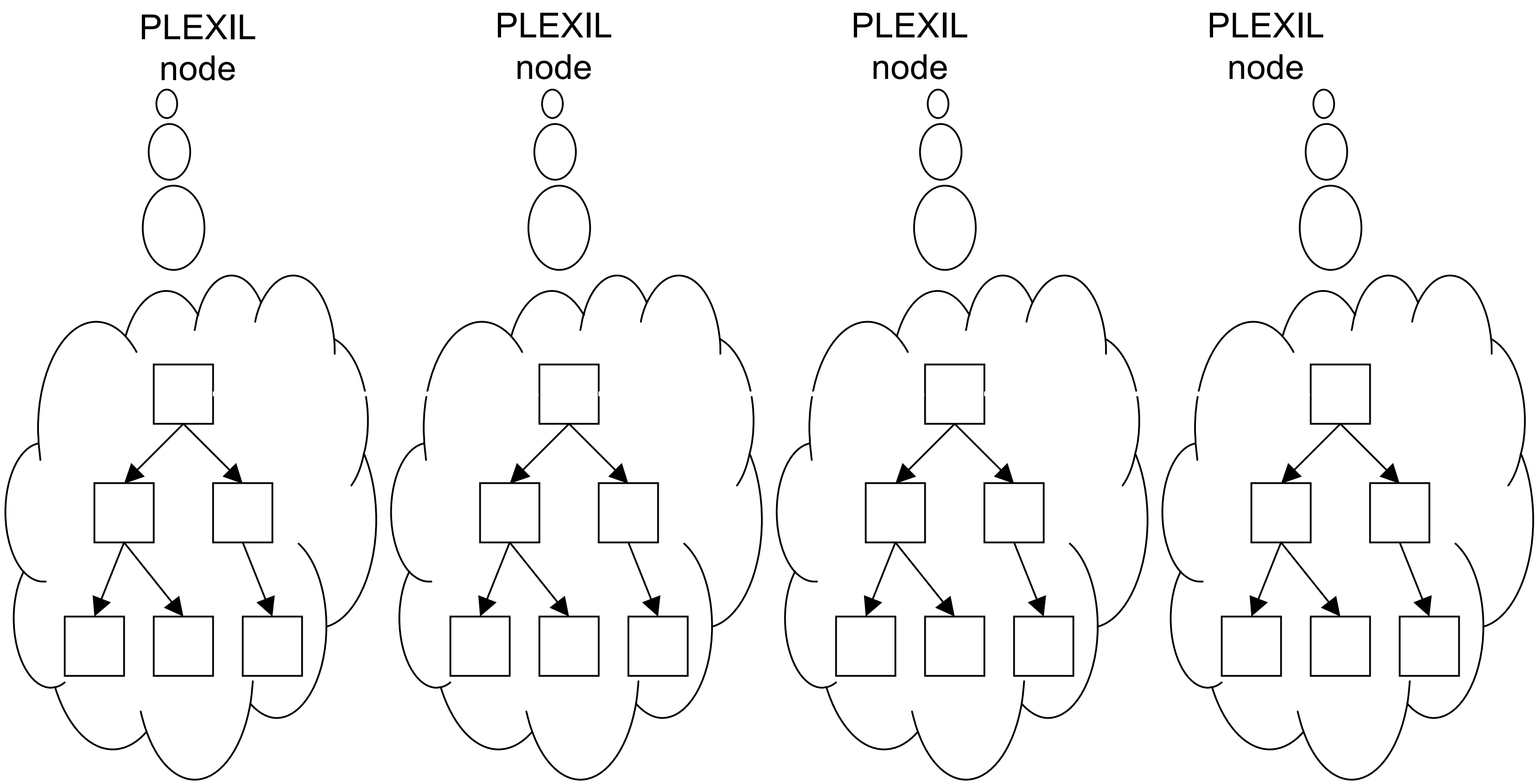
- **PLEXIL**
  - Plan Execution Interchange Language
    - For describing plans, sequences, procedures, scripts, etc.
  - Simple syntax that is very powerful
    - Timed command sequences, event driven sequences, monitors
    - Concurrent execution, repeating sequences, etc.
    - Contingencies, conditionals, etc.
  -
    - Guarantees unambiguous execution
    - Provides guarantees against deadlocks
    - Simple syntax facilitates validation and checking
  - General and reusable
- **PLEXIL is logical automation core of PRL**
  - Control logic and safety conditions in PRL map to PLEXIL
  - Execution semantics and properties of PLEXIL extend to PRL

- We investigate two ways of applying model checking to procedures
- Compositional model checking using LTSA:
  - Build Labelled Transition System Analyser (LTSA) models for
    - underlying physical system (e.g., using FSM models for simulation)
    - procedures
  - Define safety properties of interest for the procedures
  - Model check the LTSA models using compositional techniques to alleviate the state explosion problems
- SMART model checking:
  - Build SMART models of PLEXIL macros
  - Check for deadlock and behavioral correctness properties
  - Investigate scalability of the approach by defining appropriate abstractions

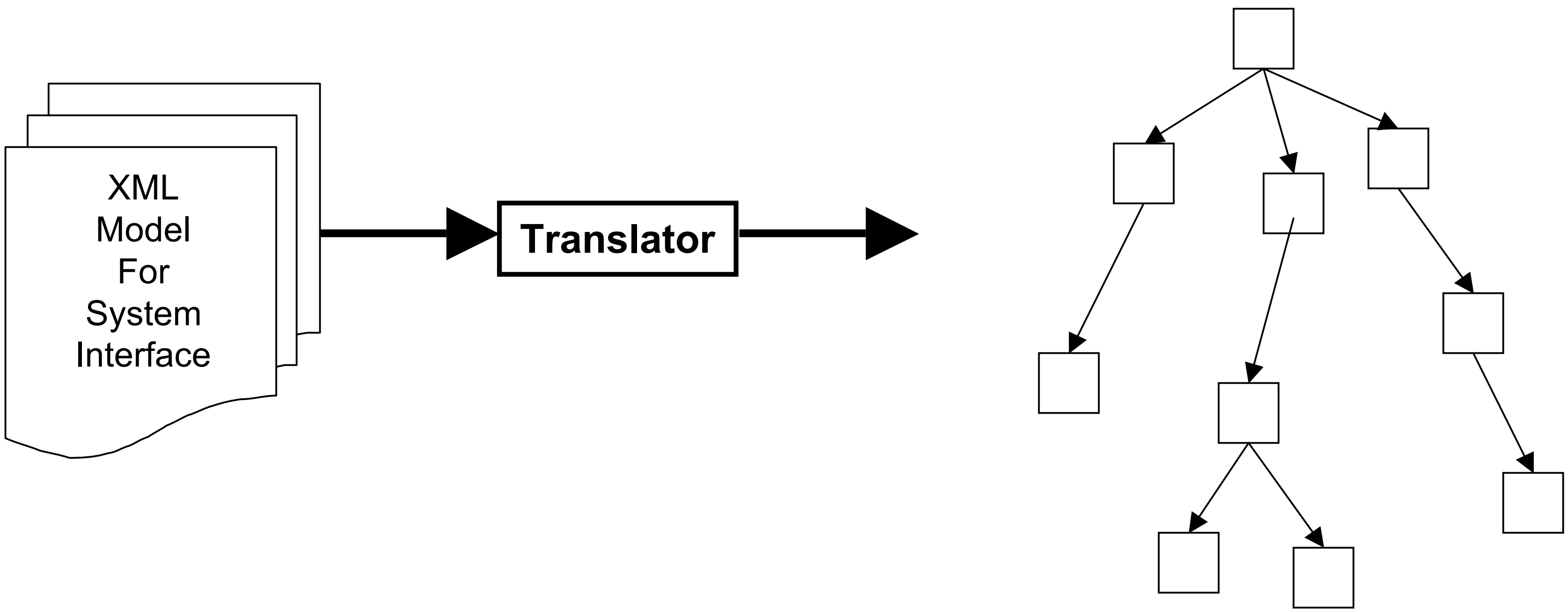
- The definition of formal semantics of PLEXIL language is necessary for the development of formal verification tools
- Our approach:
  - Described behavioral formal semantics of PLEXIL in LTSA models
    - Detection of subtle execution errors in PLEXIL models
    - Automatic translation of PLEXIL procedures into LTSA models
  - Described formal semantics of PLEXIL in PVS
    - Prove determinism and behavioral determinism for the PLEXIL language

- Behavioral model for the state *waiting* of a PLEXIL node



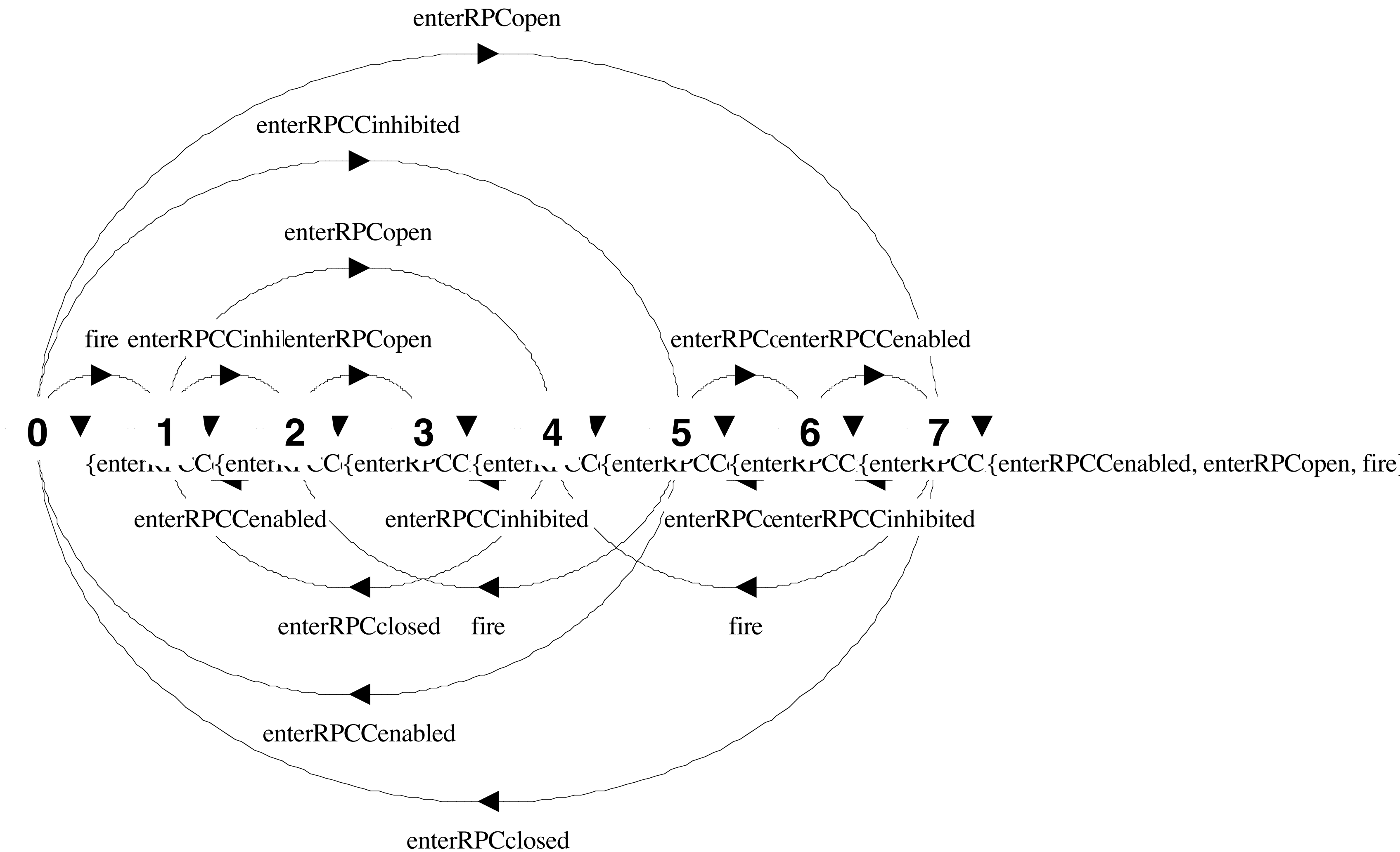


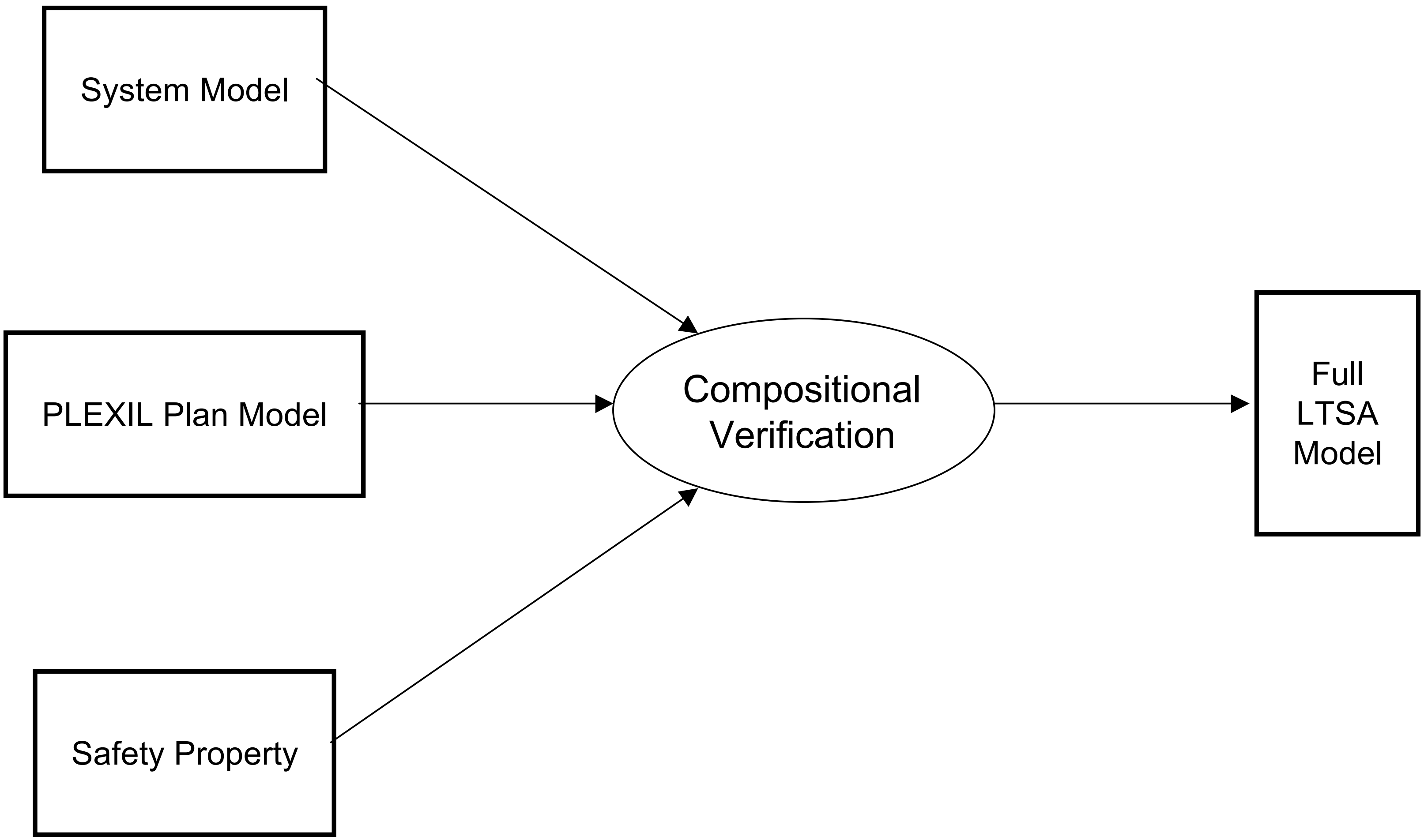
Composed LTSA Model for PLEXIL Plan

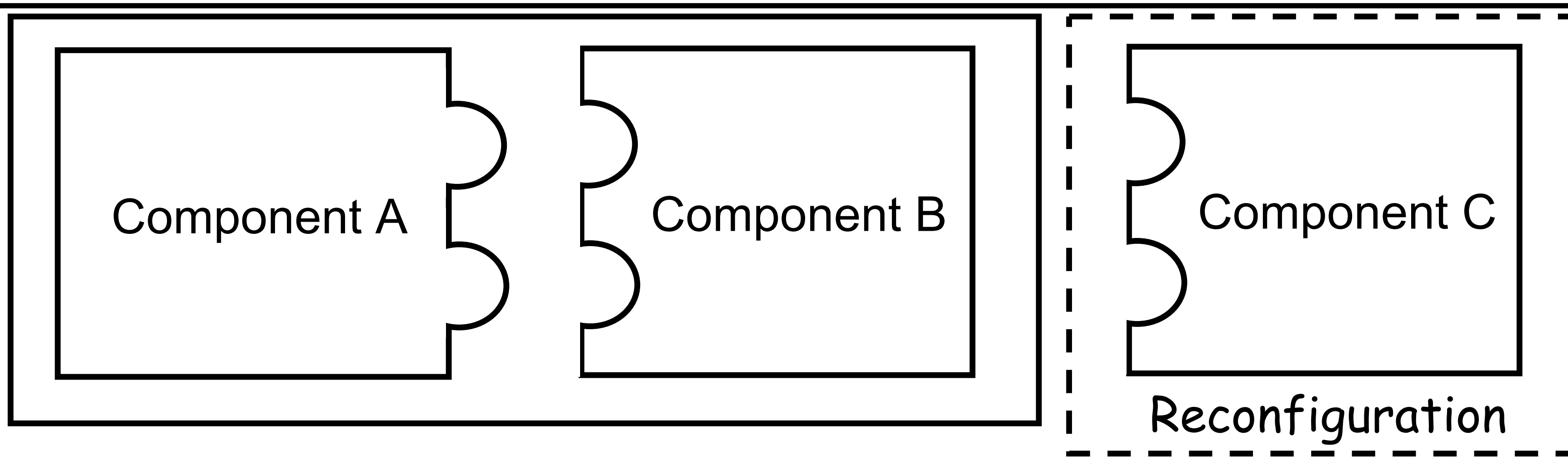


LTSA Model for System Interface

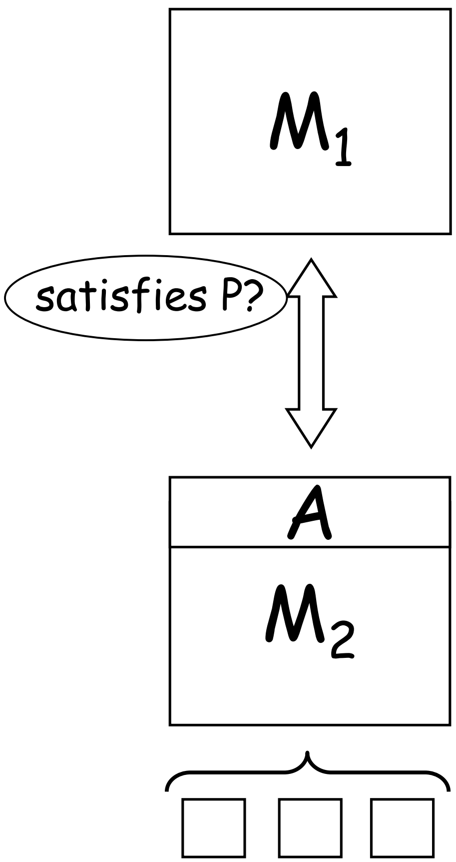
FireProof1


enterRPCopen
enterRPCCinhibited
enterRPCopen
fire
enterRPCCinhi
enterRPCopen
enterRPC
enterRPCCenabled
0
1
2
3
4
5
6
7
{enterRPCCenabled, enterRPCopen, fire}
enterRPCCenabled
enterRPCCinhibited
enterRPC
enterRPCCinhibited
enterRPCclosed
fire
fire
enterRPCCenabled
enterRPCclosed


System Model
PLEXIL Plan Model
Safety Property
Compositional Verification
Full LTSA Model

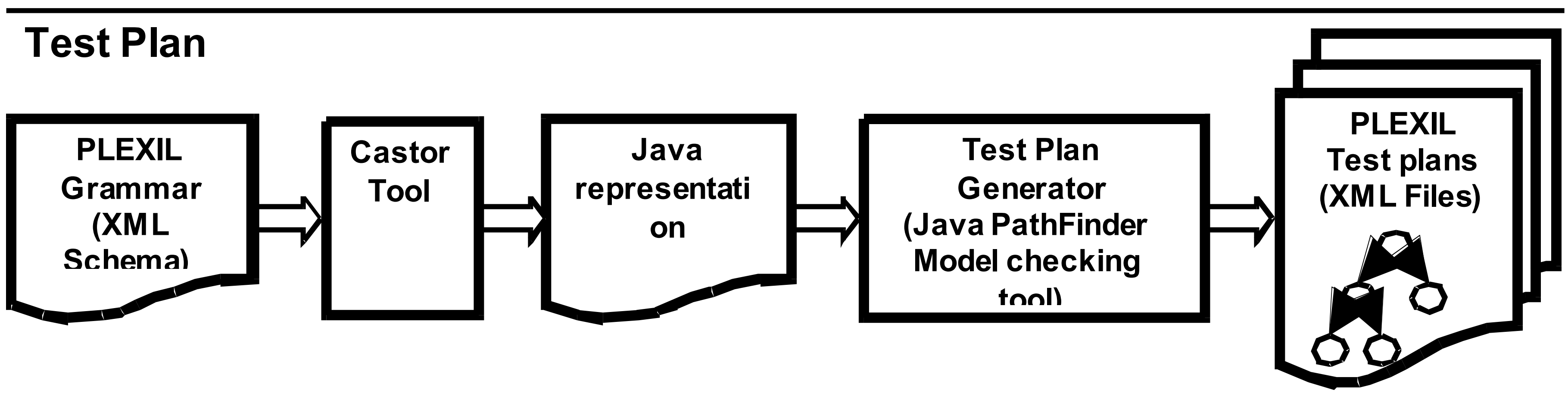


- Design-level: decompose (architecture)
  - establish contracts (assume-guarantee pairs) between components to guarantee key system-level properties
- Code-level: verify and test
  - verify or test each component against its individual contracts
- Reconfiguration
  - verify new components against contracts of substituted ones



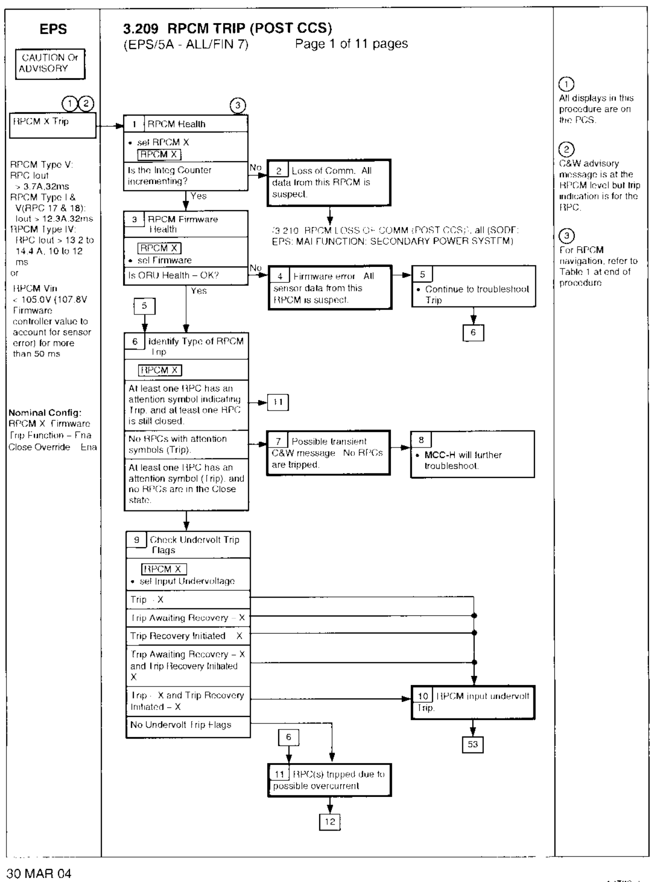
- Decompose properties of system (M_1 || M_2) in properties of its components
- Does M_1 satisfy P?
  - typically a component is designed to satisfy its requirements in *specific* contexts / environments
- Assume-guarantee reasoning: introduces assumption A representing M_1's "context"
- Simplest assume-guarantee rule

$$\frac{\begin{array}{llll} 1. & \langle A \rangle & M_1 & \langle P \rangle \\ 2. & \langle \mathit{true} \rangle & M_2 & \langle A \rangle \end{array}}{\langle \mathit{true} \rangle \; M_1 \,||\, M_2 \; \langle P \rangle}$$



- The goal is to automatically generate procedures for testing PLEXIL based on the PLEXIL grammar
  - The Castor-based translation is done
  - The test plan generation is inherited from previous research

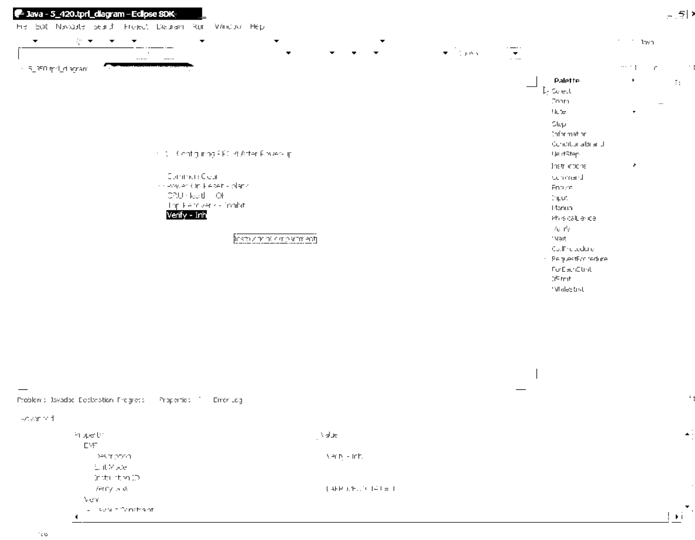
Original procedure

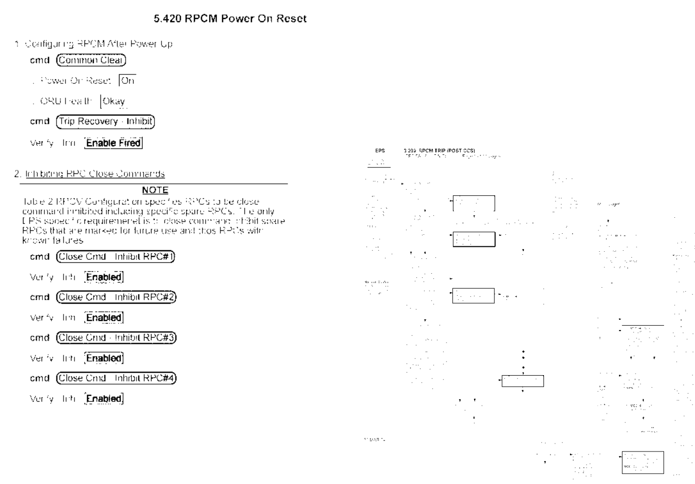


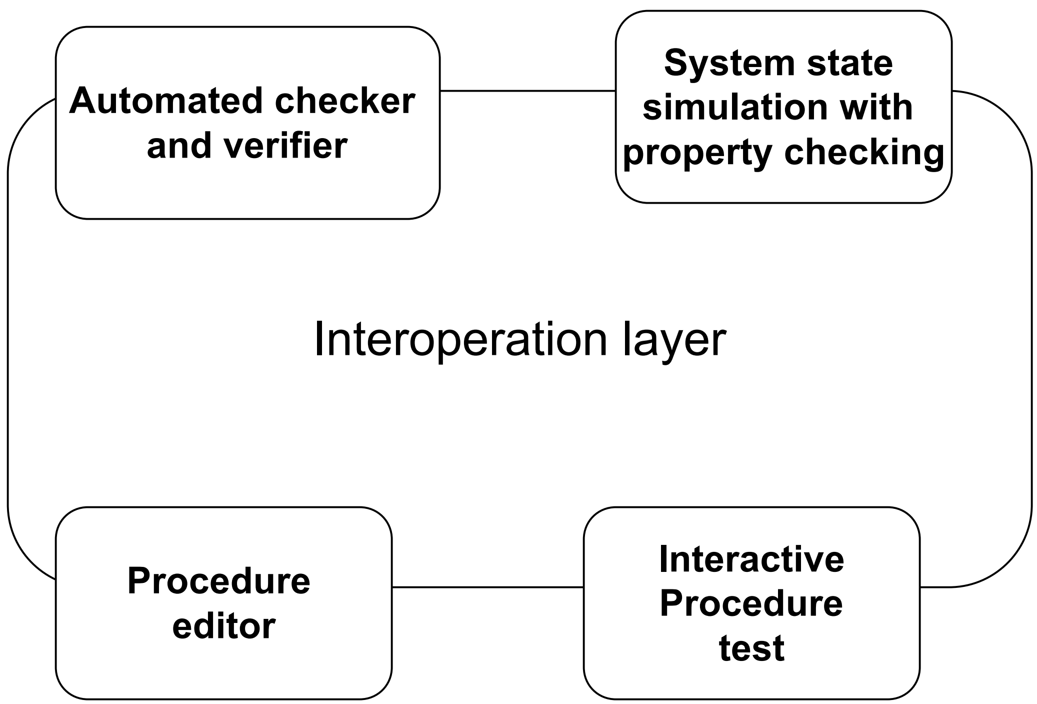
Encoding in PRL

```
<Step stepId="step3">
  <StepTitle>
   <StepNumber>3</StepNumber>
     <Text>RPCM Firmware Health</Text>
   </StepTitle>
   <InstructionBlock>
     <Instruction instructionID="step3_i1">
       <VerifyInstruction>
         <VerifyGoal>
            <TargetDescription>
               <Text>Verify ORU Health OK</Text>
            </TargetDescription>
....
```

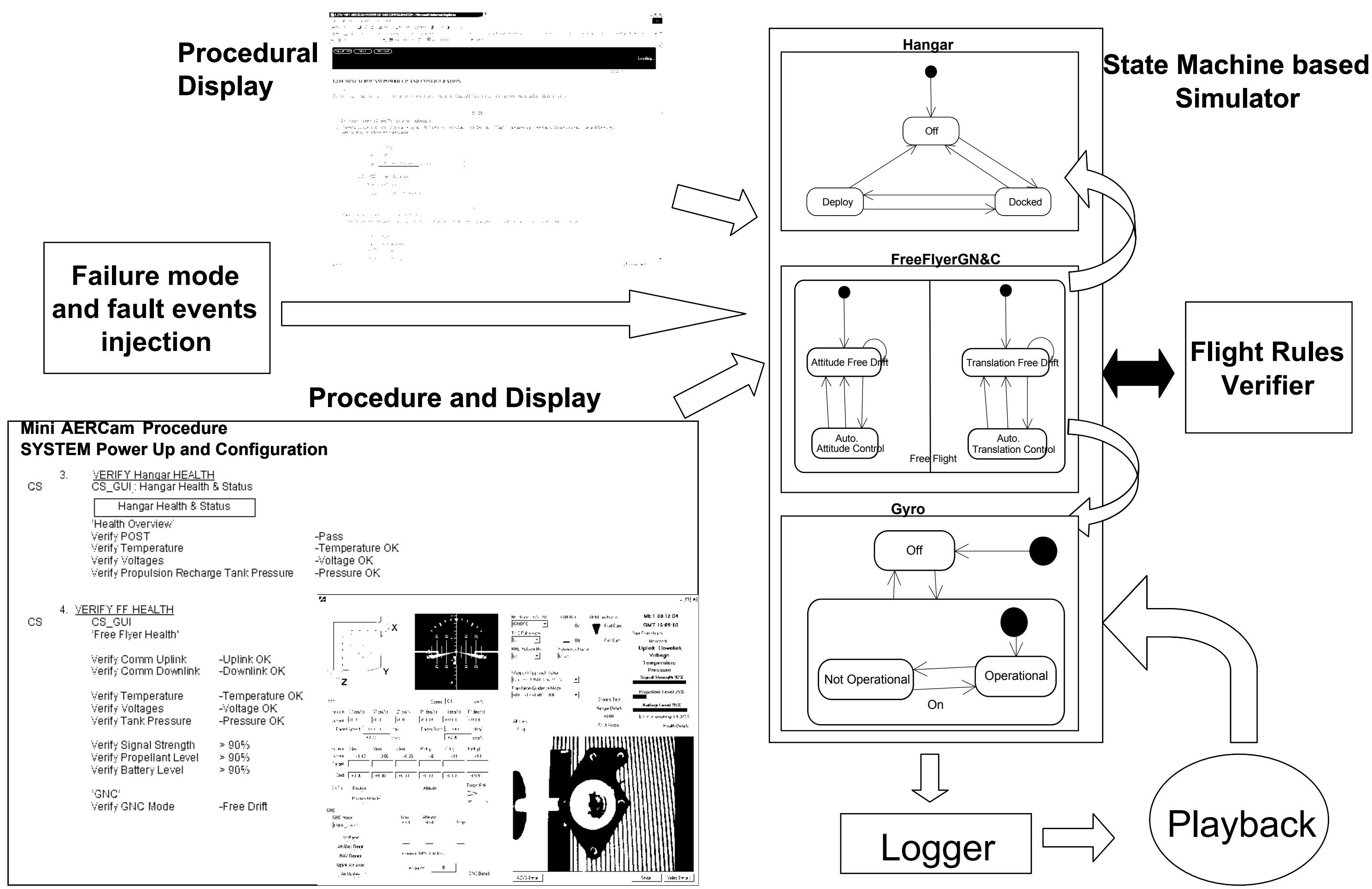
- **Authoring**
  - Graphical and Textual Editing
  - Syntax checking and Syntax constraints
- **Viewing**
  - Static and Dynamic views on procedures
- **Procedure Checking**
  - Check procedures against flight rules
  - Check procedures against constraints
  - Assist in evaluation of simulation results
  - General interface supports plug and play of validation components
- **Configuration and workflow management**
  - Support workflow, including repositories, signoffs, etc.

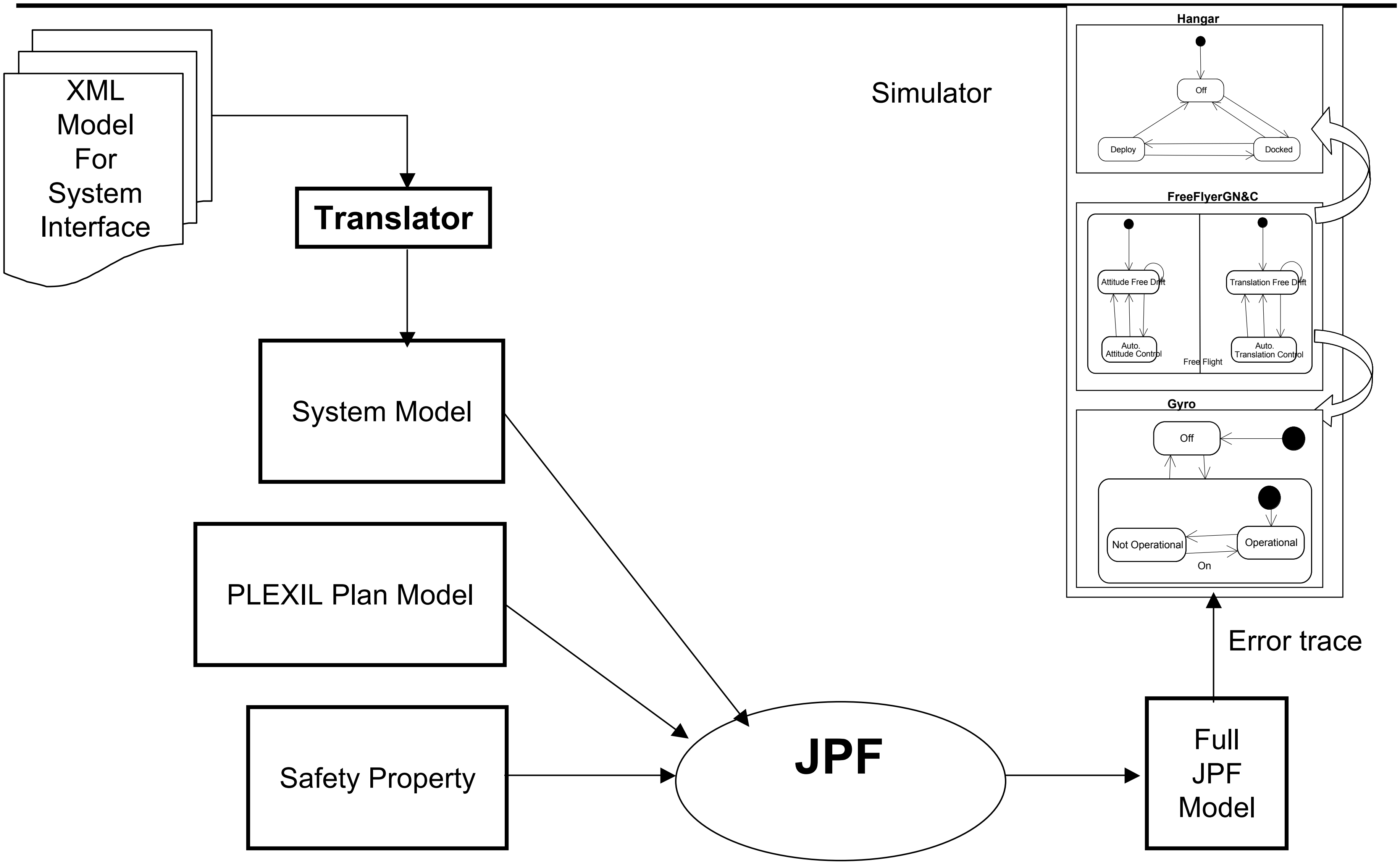


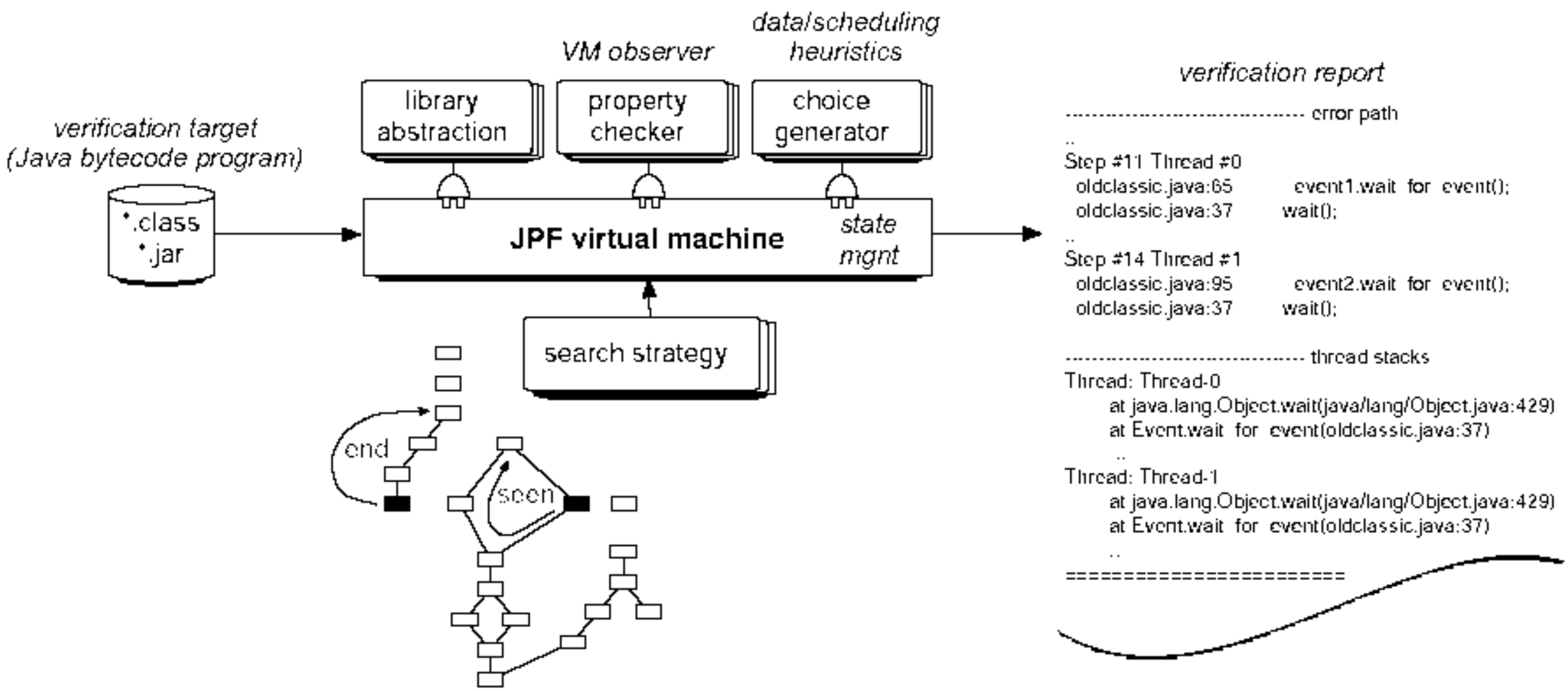



Automated checker and verifier
System state simulation with property checking
Interoperation layer
Procedure editor
Interactive Procedure test

- Build finite state machine (FSM) models describing the underlying physical system (at least, its interface to the operator world)
- Simulate the execution of the procedure in conjunction with the FSMs
- Identify missing pre-conditions for nominal state execution


Procedural Display
Failure mode and fault events injection
Procedure and Display
Mini AERCam Procedure
SYSTEM Power Up and Configuration
3. VERIFY Hangar HEALTH
CS CS_GUI: Hangar Health & Status
Hangar Health & Status
'Health Overview'
Verify POST -Pass
Verify Temperature -Temperature OK
Verify Voltages -Voltage OK
Verify Propulsion Recharge Tank Pressure -Pressure OK
4. VERIFY FF HEALTH
CS CS_GUI
'Free Flyer Health'
Verify Comm Uplink -Uplink OK
Verify Comm Downlink -Downlink OK
Verify Temperature -Temperature OK
Verify Voltages -Voltage OK
Verify Tank Pressure -Pressure OK
Verify Signal Strength > 90%
Verify Propellant Level > 90%
Verify Battery Level > 90%
'GNC'
Verify GNC Mode -Free Drift
State Machine based Simulator
Hangar
Off
Deploy
Docked
FreeFlyerGN&C
Attitude Free Drift
Auto. Attitude Control
Translation Free Drift
Auto. Translation Control
Free Flight
Gyro
Off
Not Operational
Operational
On
Flight Rules Verifier
Logger
Playback


XML Model For System Interface
Translator
System Model
PLEXIL Plan Model
Safety Property
JPF
Full JPF Model
Error trace
Simulator
Hangar
Off
Deploy
Docked
FreeFlyerGN&C
Attitude Free Drift
Auto. Attitude Control
Translation Free Drift
Auto. Translation Control
Free Flight
Gyro
Off
Not Operational
Operational
On

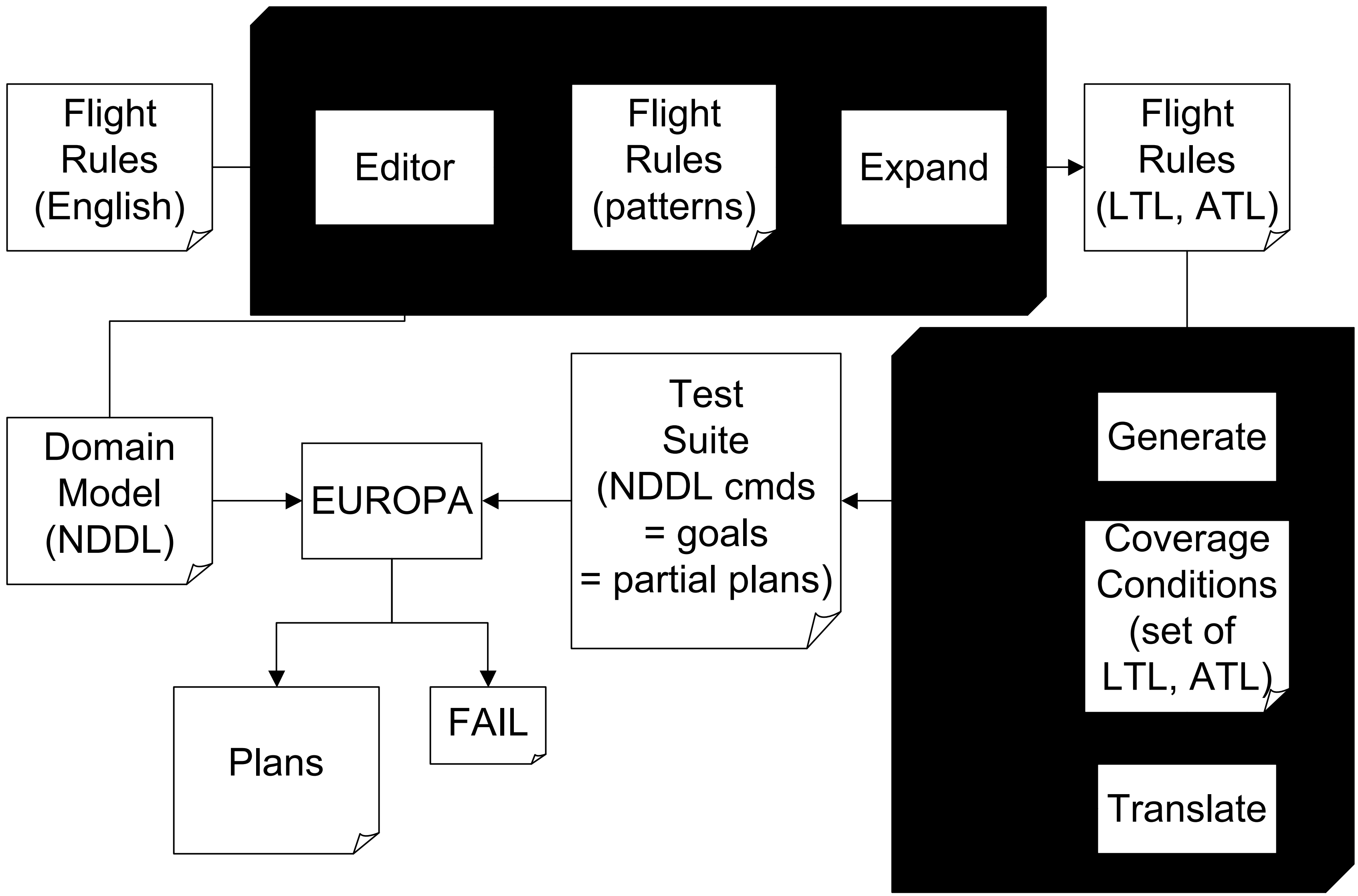


data/scheduling heuristics
VM observer
verification report
library abstraction
property checker
choice generator
verification target (Java bytecode program)
*.class
*.jar
JPF virtual machine
state mgnt
search strategy
end
seen
------------------------------------ error path
..
Step #11 Thread #0
oldclassic.java:65 event1.wait_for_event();
oldclassic.java:37 wait();
..
Step #14 Thread #1
oldclassic.java:95 event2.wait_for_event();
oldclassic.java:37 wait();
------------------------------------ thread stacks
Thread: Thread-0
at java.lang.Object.wait(java/lang/Object.java:429)
at Event.wait_for_event(oldclassic.java:37)
..
Thread: Thread-1
at java.lang.Object.wait(java/lang/Object.java:429)
at Event.wait_for_event(oldclassic.java:37)
..
========================

- Validation of planning models by translating them into model checking models
- Validation of plans and plan robustness
- Automatic generation of test cases to test against flight rules

- The goal is to study validation of planning models by translating them into SAL model checking models
- Approach:
  - Definition of a simple planning language, called APPL (A Plan Preparation Language), based on NDDL that is more amenable to formal verification
  - Automatic translation from APPL models to NDDL models
  - Automatic translation from APPL models to SAL models
    - We also study the relationship between APPL and the language unifying NDDL and Casper
  - Investigation issues of representation in SAL so that scalability problem can be avoided
    - For example, the representation of time and timers

- The goal is to automatically generate test cases for planners so that we can test against flight rules
- Process:
  - Modeling flight rules in appropriate language
    - We started with LTL (linear temporal logic), but are considering others
  - Generate coverage conditions that cover flight rules according to “unique cause” criterion
    - “Unique cause” is an extension of the commonly used MC/DC coverage criterion mandated by the FAA
  - Generate test case in the form of Europa goals (or partial plans) using the coverage conditions



Flight Rules (English)
Editor
Flight Rules (patterns)
Expand
Flight Rules (LTL, ATL)
Domain Model (NDDL)
EUROPA
Test Suite (NDDL cmds = goals = partial plans)
Generate
Coverage Conditions (set of LTL, ATL)
Translate
Plans
FAIL